Analysis of the

Crouch-Echlin Effect

Intel Corporation

Version 1.0

Acknowledgements

Intel would like to thank Mike Echlin for his time that he gave in reviewing the documents and allowing Intel access to his system for evaluation. Intel also wishes to thank all the people that provided input that resulted in this paper.

Acknowledgements 2
Observation 5
Summary 5
Intel Test Methodologies 5
Process 6
Configuration information on machines with reported observations of the Crouch-Echlin Effect. 6
Jace Crouch's "Zoom" 6
Echlin's Zenith Data System 6
Echlin's Gateway 6
Echlin System Test Methodology 7
BIOS Analysis 7
Power Cycle Testing 7
Intel Designed Motherboard Test Methodology 7
BIOS Analysis 7
Power Cycle Testing 8
External Information Analysis 8
Compaq/Digital 8
Others 8
Hypotheses and Findings 8
Hypothesis #1 (Original): Separate Code Paths for a read in the Nineteen Hundreds vs. a read in the year 2000 cause the RTC data to be corrupt in the year 2000. 8
Description 8
Findings 9
Conclusion 9
Hypothesis #2: Improper handling of the RTC's Update In Progress (UIP) bit 9
Description 9
Findings 9
Conclusion 9
Hypothesis #3: Interrupts are not locked during an INT 1A routine 10
Description 10
Findings 10
Conclusion 10
Hypothesis #4: Potential DOS kernel time/date problems 10

Description 10
Findings 10
Conclusion 10
Hypothesis #5: COMMAND.COM does not properly convert DATE/TIME strings 10
Description 10
Findings 11
Conclusion 11
Appendix A: Additional Observations 11
BIOS's 11
An anomaly with Echlin's BIOS 11
A bug in the Zenith BIOS 12
Appendix B: Yeovil's Time Dilation Testing Tools 13
Appendix C: Testing Details, Including Test Data 14
Summary of results from automated testing. 14
Automated Power Cycling 14
Intel manufactured systems 14
BIOS Analysis 14
Intel manufactured systems 14
Echlin's Zenith 286 Turbo (CPU boards #1 and #2) 15
Manual Cycling 15
Echlin's Zenith (CPU Board #2) 15
Appendix D: SSM/TSM Conversion Anomaly 18
Appendix E: Bug in Zenith INT 1A handler 19
Appendix F: Technical Descriptions of RTC/BIOS/DOS programming 20
Description of RTC Programming 20
Description of BIOS Time/Date support 20
Description of DOS kernel Time/Date support 21
Description of COMMAND.COM and Time/Date Algorithms 21

Observation

In 1997, a history professor named Jace Crouch tested his computer system for year 2000 issues. As a part of this test, the PC date was set to the year 2000. On successive boot-ups he observed that the system date would jump forward or backward. He also noted that other parts of the system that are controlled by the CMOS (e.g., the number of serial ports) were malfunctioning. Crouch reported these findings on the Internet.

A computer programmer named Mike Echlin read the findings and attempted to duplicate them. Echlin observed similar problems to those reported by Crouch on four systems he tested: a 286-, two 386-, and a 486-33 processor-based PCs. This effect of dates jumping forward or backward on successive boot-ups of systems with clocks set after the year 2000 is now known as "Time Dilation" or the "Crouch-Echlin" Effect. Known acronyms are CE, TD, and CE-TD. This paper will refer to the Crouch-Echlin Effect or the Effect.

As described by Mr. Echlin on his web site:

On boot up, after year 2000, the computer may occasionally make a jump in its date or time. For example, you have been running your computer every day since January 1, 2000. It is now January 9. You have found and fixed most of the important Year 2000 problems you have seen. Yesterday when you shut off your computer the date and time were correct. Today you start your computer, but it reports the date as April 17, 2000, and the time is correct.

Mr. Echlin hypothesized, *"These time and date instabilities can occur after the year 1999 because the BIOS then takes longer to access and process data obtained from the RTC (Real Time Clock), and on systems with a non-buffered RTC the BIOS may do this while the data is incorrect. If this occurs during POST (Power On Self Test), the incorrect data from the RTC may be used to calculate the time and date for the software clock, resulting in an inaccurate time, or date, or both. Further, this incorrect time or date may get written back to the RTC/CMOS, thereby sustaining the time and date errors until the RTC/CMOS is reset by the user or by remediative software."*

Testers at Digital Equipment Corporation reported that they independently observed behaviors similar to the Crouch-Echlin Effect.

Summary

Intel has not observed the Crouch-Echlin Effect as reported by Jace Crouch and described by Mike Echlin. We have done an extensive investigation into this issue and have come up with no evidence supporting this Effect. The tests we have performed include automated power cycling, system and code analysis, and manual reproduction techniques as described by Mr. Echlin. We have tested for this Effect on Intel manufactured systems, from Intel's original 301 design to its latest motherboards and chipsets, and we have not observed the Effect on any of them. Nor have we detected the Effect on systems reported to have experienced it.

Separate code paths based on the century clearly are not the cause of the reported Crouch-Echlin Effect, since there is no century logic in the BIOS of machines reported to have this Effect. Improper handling of the UIP bit also is not the cause, since machines with the reported effect do handle it properly. Although Compaq identified a clone in which the UIP bit was not properly handled, the company still could not reproduce the Effect. The implication of non-buffered Real Time Clocks is an artifact of a disproved hypothesis and, therefore, not directly related to the observations. Double-buffered RTCs may hide some of the effects of an out-of-specification BIOS, thus leading the observer to believe, erroneously, that the problem lies in the existence of the buffer, or lack of one.

Intel advises that users check the year 2000 capabilities of their PCs by using one of the reputable utilities now on the market such as National Software Testing Laboratories' YMARK2000. Intel also recommends that users contact the manufacturer of their system for the latest BIOS upgrade if concerned about the quality of their BIOS.

Process

Intel attempted to duplicate the issue described in the Crouch-Echlin TD white paper written by Mr. Echlin, using Intel designed motherboards. We also attempted to duplicate the observations with machines on which the Effect was reportedly observed, such as those supplied to us by Mr. Echlin. Tests included automated external source power cycling and manual power cycling.

We started with Mr. Echlin's original hypothesis (see below) to explain this anomaly, then developed and tested several additional hypotheses about what might cause the Effect. The tests we performed include code analysis to determine century-based logic and improper handling of the RTC's Update In Progress (UIP) bit. In addition, we have been working with experts at Phoenix (BIOS vendor) and Compaq Computer Corporation (PC manufacturer) to identify and test potential causes of an anomaly such as Crouch-Echlin.

Configuration information on machines with reported observations of the Crouch-Echlin Effect.

Jace Crouch's "Zoom"

286-12 Clone

BIOS	Award 4.50g
CONFIG.SYS	unknown
AUTOEXEC.BAT	unknown

Intel has not seen this machine.
Note: The BIOS on this machine is described as an Award 4.50g ***and*** a C&T SCAT BIOS on Echlin's website. These are not the same BIOS and we believe the C&T SCAT BIOS information to be in error.

Echlin's Zenith Data System

Zenith Data Systems, with two separate sets of Processor-I/O cards provided. This is a single chassis with a passive backplane. Mr. Echlin sent us two sets of CPU/IO boards for this machine. Each set contains one CPU board with BIOS chips and one I/O board containing the RTC.

Zenith 286-12 'Turbo'

BIOS #1	Zenith p/n 444-424-13 / 444-423-13
BIOS #2	Zenith p/n 444-424-6
CONFIG.SYS	none
AUTOEXEC.BAT	(as shipped) PROMPT DT TDFIX.EXE

Note: no version information included in BIOS code

Echlin's Gateway

Gateway 486-33

BIOS	Phoenix v010D22-2 RIDE
CONFIG.SYS	unknown
AUTOEXEC.BAT	unknown

This is Mr. Echlin's server and boots into Windows 95. Intel has not seen this machine.

Echlin System Test Methodology

Intel has attempted to duplicate the Crouch-Echlin Effect on one of the systems that Mr. Echlin has observed to exhibit this anomaly. These are the steps we took reproduce and analyze the Effect.

The system Mr. Echlin sent us is a passive backplane, 80286-based design by Zenith Data Systems. Two separate sets of Processor - I/O cards were included.

The RTC used in this system is the Motorola 146818 component. This component meets Mr. Echlin's definition of an "unbuffered" RTC.

BIOS Analysis

BIOS source code was not available, so a complete analysis of the BIOS was not possible. We did observe that there were different BIOS versions on the two processor cards. To analyze the BIOS code on Mr. Echlin's boards we single stepped through the BIOS code with a debugger and disassembled the object code.

Power Cycle Testing

A series of "fast" automated power cycle tests were conducted with the system. Processor Card #1 was tested for 88 hrs and 3100 power cycles, Processor Card #2 was tested for 15 hours and 600 power cycles. The time and date were logged for each power cycle and compared to a base clock to look for any unexpected jumps. Mr. Echlin's utility TDFIX.EXE (see Appendix A) was run on this system during these power cycle tests. We have also attempted to duplicate the problem through a manual power cycling process as described in Mr. Echlin's white paper. The system manually power cycled 42 times on 16 days over a 5 week period.

Intel Designed Motherboard Test Methodology

Intel has attempted to duplicate the Crouch-Echlin Effect on Intel manufactured systems through a series of automated and manual tests, code reviews and logic analysis. These are the steps we took to verify the hypothesis put forward by Mr. Echlin.

BIOS Analysis

Intel conducted a code review of the BIOS cores used on our Intel motherboards from our earliest 80386-based motherboards to our latest Pentium® II processor-based motherboards. The primary hypothesis for the cause of the Crouch-Echlin Effect is that the BIOS incorrectly accesses the RTC during an update cycle to the clock registers. The code review concentrated on three different aspects that could contribute to this assumed cause of the Crouch-Echlin Effect.

1. UIP bit: The UIP (Update In Progress) flag is checked by INT 1A handlers (such as the INT 1A handler in the BIOS) or other software that reads the RTC to determine if an update to the clock registers is currently in progress. If an update is in progress, the software must wait to read the clock registers, otherwise incorrect information may be read. If the UIP bit is cleared, the software has a maximum of 244 microseconds to complete the read of the RTC register before an update cycles will occur. The software must ensure it completes the read cycle before this 244 microsecond period expires.

2. Interrupts: Interrupts must be disabled before checking the UIP bit and while accessing the RTC registers. This must be done to ensure an interrupt service routine is not executed after the UIP bit is checked but before the RTC register is accessed, which would cause the 244 microsecond period to be exceeded. After the RTC access cycle is complete, the software must re-enable interrupts.

3. Century value code logic: It has been stated that the Crouch-Echlin Effect only occurs after the century byte has been changed from 19 to 20. Mr. Echlin has hypothesized that this is because the BIOS code executes an RTC access differently based on the value of the century byte.

Power Cycle Testing

Intel conducted a series of power cycle tests on a representative sample of four Intel designed motherboards that use different BIOS cores and different RTC controllers. These tests were conducted using an automated power cycling test station. This test station cycle timing is totally independent of the system clocks of the units being tested.

The first set of power cycle tests were set for "fast" cycles, approximately one minute on, one minute off (2 minute cycle). The time and date were logged for each power cycle and compared to a base clock to look for any unexpected jumps. This series of tests ran for over 135 hours and over 4000 power cycles.

The second set of power cycle tests were set for "slow" cycles, approximately one hour on, one hour off. The time and date were logged for each power cycle and compared to a base clock to look for any unexpected jumps. This series of tests ran for 90 hours and 46 power cycles.

External Information Analysis

We have attempted to research many of the occurrences of the Crouch-Echlin Effect and correlate the findings. The objective is to track down the reported observations, get specific configuration information, and determine exact tests run. This is ongoing process. Following is a summary of Intel's efforts to date.

Compaq/Digital

Compaq received one of the computers on which the Effect was observed. Compaq has not been able to reproduce the Effect either through automated or manual tests. It did discover a "mis-matched" BIOS/RTC where the BIOS does not honor the UIP bit and the RTC is not buffered. Compaq received the machine without the original hard drive so it is impossible to know whether the exact configuration was successfully duplicated.

Compaq reported that they received one of the computers on which the Effect was observed. Compaq reported that it has not been able to reproduce the Effect either through automated or manual tests. It did reportedly discover a "mis-matched" BIOS/RTC where the BIOS does not honor the UIP bit and the RTC is not buffered. Compaq reported that it received the machine without the original hard drive so it is impossible to know whether the exact configuration was successfully duplicated.

Others

We have attempted to procure systems on which the Effect has been observed by contacting individuals who have reported they have seen the Effect. Other than the machines listed above, we have been unsuccessful in procuring these systems. Several individuals we contacted said that they were misquoted in the press and have not actually seen the Effect, but remain cautious as to whether the Crouch-Echlin Effect exists.

Hypotheses and Findings

Hypothesis #1 (Original): Separate Code Paths for reading the RTC before and after the transition to the year 2000 cause the RTC data to be corrupt.

Description

Mr. Echlin hypothesized that the Crouch-Echlin Effect was caused by a BIOS choosing a new execution path in the year 2000 and beyond, which would incorrectly access the system's Real Time Clock (RTC). The hypothesis suggests that there are different code/logic paths in the BIOS during a "read" in the 1900s, vs. a read in the year 2000 time frame. This difference might cause a longer path to be active in the year 2000 and beyond, and might violate the 244µs "OK to read" specification of the RTC's Update In Progress flag (the UIP bit). The extra time this would take could corrupt CMOS registers where date and time information is kept.

Findings
We have reviewed the BIOS code on our own systems from our earliest 386-based motherboards to our latest designs and have found no logic based on the century during RTC access sequences. Phoenix, a major motherboard BIOS vendor to Intel, and Compaq engineers have reviewed our findings and concur. No decision logic based on time or date values is performed during RTC reads and writes. In essence, the BIOS checks the UIP bit and, if OK to proceed, reads the RTC registers and stores the values. Then, independent of the RTC, the BIOS determines if the century byte needs to be updated and, if it does, writes the new data back to the century byte in the CMOS memory. On Echlin's Zenith boards this century-update-algorithm doesn't exist; the century must be manually updated via the DOS DATE command (i.e. these boards fail the century rollover test). On newer systems, this algorithm exists in both the POST routine and INT 1A routine.

We used the DOS "Debug" utility, an EPROM programmer/reader, and a hex code disassembler to read the actual BIOS code on the Echlin machines, and the logic paths *are not different* based on the value of the century byte. Also, we observed that the BIOS spends only 35 microseconds while reading or writing RTC registers on Echlin's 286-12.

Note: This theoretical 'year 2000 code path,' if applied to Pentium processor-based computers, would have to be 20 times longer than the theoretical 'nineteen hundreds code path' in order to exceed the 244µs limit due to the increase in processor speed between the 286 processor and the Pentium processor.

Conclusion
This hypothesis is disproved. The original hypothesis of separate code paths based on the century remains unsupported by the evidence.

Hypothesis #2: Improper handling of the RTC's Update In Progress (UIP) bit

Description
The UIP (Update In Progress) flag is used by applications to determine if an update to the clock registers is currently in progress. If an update is in progress, the application must wait to read the clock registers, otherwise incorrect information may be read. If the UIP bit is cleared, the application has a minimum of 244 microseconds to complete the read of the RTC register before an update cycle will occur. The application must ensure it completes the read cycle before this 244us period expires.

This hypothesis, proposed by Intel and Compaq engineers, suggests that the RTC properly sets the UIP bit, but BIOS code in violation of specifications ignores it and tries to read the RTC registers anyway. This would explain why the Effect is not seen on machines with double-buffered RTCs because the data is held in a buffer independent of the RTC registers, even though the BIOS is still in violation of the spec.

Findings
We reviewed the BIOS code on Echlin's boards and found that the BIOS properly honors the status of the UIP bit. We also reviewed the code on Intel manufactured motherboards and found that BIOS honors the UIP bit on these systems as well.

Conclusion
This argument is disproved. The UIP bit is handled properly.

Note: Compaq suggests that a BIOS that was written for a double-buffered RTC could have been installed on some systems with non-double buffered RTCs. The company has a no-name clone machine that shows this mismatch, but has not been able to reproduce the Crouch-Echlin Effect on this machine through automated or manual tests. This is one of the machines on which Digital reportedly observed the Effect.

Hypothesis #3: Interrupts are not locked during an INT 1A routine

Description

If a program executes more than 244 microseconds worth of code between the time that it checks the "update-in-progress" flag and the time it completes its interactions with the RTC, then its interaction may malfunction (i.e. the value read/written may be random/ignored). If a program fails to lock interrupts across the whole operation of checking the "update-in-progress" flag and the completion of its interaction with the RTC, then interrupt handlers might execute, which would extend the operation beyond the 244 microsecond limit and thus cause the interaction to malfunction.

Findings

We have determined that the INT 1A handler on every system we tested, including Mr. Echlin's Zenith, properly locks interrupts while performing its duties.

Conclusion

This hypothesis is disproved.

Hypothesis #4: Potential DOS kernel time/date problems

Description

The DOS kernel programs the RTC Status Register A and Status Register B values based upon industry-standard definitions of those values. If DOS were being loaded on a machine with an RTC that kept incompatible information in those status registers, then DOS might cause the RTC to malfunction.

Findings

We investigated the date and time handling of DOS versions since 3.30 including MS-DOS, PC-DOS, and DR DOS. The DOS kernel's time-keeping logic is not affected by the date (including the century). The DOS kernel's date-keeping logic is not affected by the century – it only divides or multiplies by 100 to convert from a year (e.g. 1998) to a century and year (e.g. 19 and 98) and back.

DOS' reliance on the BIOS' Get System Time function to tell it when the date has changed can cause DOS to lose track of days passing. First, most versions of DOS (and most BIOS's) consider the "midnight has passed" value to be a flag (i.e. yes or no) rather than an integer value (i.e. the number of midnights which have passed). Thus, when the BIOS tells DOS that a midnight has passed, DOS increments its internal "days since January 1st 1980" variable by one day, even if more than one midnight has passed between DOS calls to the Get System Time function. This is a common problem for people who leave their machines on, but idle, over the weekend and have DOS report the wrong date when they return to work on Monday. A second problem is that the BIOS resets its "midnight has passed" variable after each Get System Time call. This means that if some piece of software other than DOS calls the BIOS to get the System Time, then DOS may not be the one to receive notification that midnight has passed. These two problems have nothing to do with year 2000 issues and have existed since the first PC.

Conclusion

We found only the harmless discrepancy between times reported by DOS and times reported by the BIOS (i.e. the DOS time and RTC times differing by as much as 1 second every 2800 seconds on Mr. Echlin's system). We found no other problems related to time/date issues in any version of the DOS kernel as far back as PC-DOS 3.30.

Hypothesis #5: COMMAND.COM does not properly convert DATE/TIME strings

Description

COMMAND.COM must convert from date and time values returned by the DOS kernel to date and time strings displayed to the user. Conversely, COMMAND.COM must convert from date and time strings entered by the user to date and time values passed to the DOS kernel.

Findings
We looked at the DATE/TIME conversion routines used by COMMAND.COM . We did not find any time/date errors in any versions of COMMAND.COM shipped with MS-DOS 3.30 and later. We have not checked for errors in command processors shipped by Digital Research or as third-party add-ons.

Conclusion
This hypothesis is disproved.

Appendix A: Additional Observations

BIOS's

An anomaly with Echlin's BIOS
While setting the system up for power cycle testing on one of Mr. Echlin's Zenith boards, we discovered a discrepancy between the timer tick conversion algorithm used in the BIOS POST routine and the conversion algorithm used by DOS. The BIOS uses an algorithm to convert the hours/minutes/seconds since midnight (as reported by the RTC) to timer ticks since midnight and stores the result in the BIOS data area (BDA) at offset locations 6CH-6FH. This counter then gets incremented every time the INT 08H system timer interrupt occurs (approximately 18.2 times per second). At midnight, a rollover flag gets set (offset location 70H), and the counter starts over at zero. DOS' TIME command doesn't actually go to the RTC to get the time. Instead, it reads this timer tick counter in the BDA and then converts these timer ticks back to seconds/hours/minutes-since-midnight.

On Processor Card #2, TDFIX.EXE was observed to intermittently detect a difference between the DOS system clock and the RTC. Upon further investigation we discovered that the algorithm used by the BIOS version on Processor Card #2 to convert the time from the RTC to the number of timer clock ticks since midnight lost approximately 1 second every 2800 seconds. This would account for the zero to 30-second discrepancy between the DOS system clock and the RTC, depending upon what time during the day the system was booted. This discrepancy matched the value reported by TDFIX.EXE. This conversion error occurs independent of the value of the century byte, and occurs on every boot cycle. For a more detailed explanation, please see the Appendix C.

The BIOS executes its conversion routine only at POST, not during an Interrupt 1A routine. DOS converts from ticks to seconds whenever it displays the time via the PROMPT command or TIME command or whenever it creates or modifies a file (to set the file time). DOS converts from seconds to ticks whenever it is called to set a new time. If the BIOS uses a different conversion routine than does DOS, then the time according to DOS would be different from the time contained in the RTC. On this particular BIOS, the discrepancy was unusually large. For example, with Processor Card #2, if the computer was powered on

at 12:47 AM the timer ticks in the BDA would be slow by 1 second;

at 1:34 AM the timer ticks in the BDA would be slow by 2 seconds;

at 2:21 AM the timer ticks in the BDA would be slow by 3 seconds;

at 3:08 AM the timer ticks in the BDA would be slow by 4 seconds;

at 3:54 AM the timer ticks in the BDA would be slow by 5 seconds;

at 4:41 AM the timer ticks in the BDA would be slow by 6 seconds;

at 5:28 AM the timer ticks in the BDA would be slow by 7 seconds;

at 6:14 AM the timer ticks in the BDA would be slow by 8 seconds;

at 7:01 AM the timer ticks in the BDA would be slow by 9 seconds;

at 11:24 PM the timer ticks in the BDA would be slow by 30 seconds.

Please see Appendix C for a more technical description of this anomaly.

The discrepancy that Mr. Echlin's TDFIX utility detected on this machine is explained by these conversion algorithms.

We have not examined the conversion algorithms of non-DOS operating systems, but we expect that they use different algorithms than DOS does and will, therefore, have slightly different discrepancies with the BIOS. All of these discrepancies are small, have existed since the first PC, are unrelated to year 2000 issues, and are likely to trigger only those programs designed to look for a difference between the DOS system clock and the RTC.

A bug in the Zenith BIOS

We found one bug in the interrupt INT 1A handler on Mr. Echlin's Zenith system. In a very rare instant in time, it is possible for the INT 1A handler to bypass the check for the UIP bit status. For a complete description of this bug, please see Appendix E.

This Zenith BIOS bug is not related to year 2000 issues, and we do not believe that this Zenith BIOS bug explains any of the Crouch-Echlin Effects that have been reported by Mr. Echlin and others.

Appendix B: Yeovil's Time Dilation Testing Tools

Yeovil Systems Research and Development (Mr. Echlin's company) created a toolkit to diagnose and fix the Crouch-Echlin Effect. In the process of our testing, we did use this toolkit. First is the TDTEST.EXE application, which is the diagnostic component of the toolkit and second, is the TDFIX.EXE application that is designed to fix the "Crouch-Echlin Effect." Since we were unable to duplicate the Effect, we cannot confirm or deny the toolkit's functionality in regard to detecting or fixing the Effect.

We did, however, examine the code and found that both the TDTEST and TDFIX utilities leave interrupts enabled (we understand that this problem has been fixed in a subsequent release of the utility). This would allow an interrupt to occur which would delay the completion of the programming of the RTC beyond its 244 microsecond "grace period" and into the update period during which RTC programming can produce invalid results. It also means that an interrupt could occur which executes an interrupt handler, which then interacts with the CMOS ports, thus changing the CMOS index being read/written by this program. This latter problem occurs frequently when TDTEST is run from within a Windows DOS Prompt window. In addition, we found that neither program contains a CLI instruction except around some stack-switching code that is part of the C runtime environment. Mike Echlin has released a newer version that reportedly corrects this problem. Intel has not tested it.

We ran TDFIX. TDFIX reported that it found the Crouch-Echlin Effect symptom and that TDFIX fixed it. It appears that the utility is reporting discrepancies between RTC data and BIOS data, as on Mr. Echlin's Zenith. While this discrepancy exists every time the machine is powered on TDFIX does not consistently detect it.

Appendix C: Testing Details, Including Test Data

Summary of results from automated testing.

Platform	ZDS 286 (CPU1, I/O1)	ZDS 286 (CPU2, I/O1)	301	386H	C&T SCAT	AL440LX
Config Details	Proc. = 80286 Speed = 8MHz BIOS = 444-424-13 444-423-13	Proc. = 80286 Speed = 8MHz BIOS = 444-424-6	Proc. = 386DX Speed = 16MHz BIOS = 1.10.04.A0	Proc. = 386DX Speed = 33MHz BIOS = 1.10 34.O1	Proc. = 386SX Speed = 20MHz BIOS = SCATsx/PEAKsx 3.0.4	Proc. = Pentium® II Speed = 266MHz BIOS = 4A4LL0X0.86A.0028.P10
RTC controller	Mot. 146818	Mot. 146818	Mot. 146818	Integrated in chipset	Integrated in chipset	Integrated in 82371AB (PIIX4)
TDTEST.EXE (Echlin's test)	Non-buffered RTC	Non-buffered RTC	Non-buffered RTC	Non-buffered RTC	Non-buffered RTC	Non-buffered RTC
Automated Power cycling (fast)	Test time = 88 hrs Power cycles = 3108 Result = no failures	Test time = 15 hrs Power cycles = 600 Result = no failures	Test time = 139 hrs Power cycles = 4026 Result = no failures	Test time = 139 hrs Power cycles = 4026 Result = no failures	Test time = 139 hrs Power cycles = 4026 Result = no failures	Test time = 139 hrs Power cycles = 4026 Result = no failures
Automated Power cycling (slow)			Test time = 90 hrs Power cycles = 46 Result = no failures	Test time = 90 hrs Power cycles = 46 Result = no failures	Test time = 90 hrs Power cycles = 46 Result = no failures	Test time = 90 hrs Power cycles = 46 Result = no failures

Automated Power Cycling

Intel manufactured systems
No case of the Crouch-Echlin Effect was observed. See summary above.

BIOS Analysis

Intel manufactured systems
On Intel manufactured systems, our investigations revealed the following facts:

1. UIP bit: We confirmed all our BIOS cores utilize the UIP bit correctly.

2. Interrupts: We confirmed that all of our BIOS cores correctly disable/enable interrupts during RTC accesses.

3. Century value code logic: We confirmed all of our BIOS cores access the RTC using the same code sequence independent of the value of the century byte. Part of Mr. Echlin's theory was that something somewhere MUST utilize the century byte, since Leap year in 2000, for example, is different from 1900. Mr. Echlin assumed this was done on the fly. Actually, these calculations are done at a higher level, after low-level RTC access is completed.

Echlin's Zenith 286 Turbo (CPU boards #1 and #2)

1. UIP bit: Both BIOS versions utilize the UIP bit correctly.

2. Interrupts: We confirmed both BIOS versions correctly disable/enable interrupts during RTC accesses.

3. Century value code logic: We confirmed both BIOS versions access the RTC using the same code sequence independent of the value of the century byte.

On CPU board #1 we did not observe any case of the Crouch-Echlin Effect, nor did TDFIX.EXE detect any discrepancy between the DOS system clock and the RTC. We also did not observe any case of the Crouch-Echlin Effect on CPU board #2; however, we did see some anomalies with the BIOS on this board. Please see "Additional Observations" for more information on these anomalies.

Manual Cycling

Echlin's Zenith (CPU Board #2)

Mr. Echlin's machine was set up with only a PROMPT command in the AUTOEXEC.BAT and no CONFIG.SYS. The system was powered on and off manually at random times throughout the day, every day, for two weeks. This was done to duplicate the test procedure as described by Mr. Echlin. A log was kept of the power on and power off times as well as the date and time displayed by the system. A single, independent clock was used throughout the experiment to compare with the system time. See actual data below.

No jumps or anomalies in the date or time displayed by the system were observed. The minimum observable unit of measure is one minute since the clock used was an analog wristwatch.

System: Echlin's Zenith 286-12

Configuration: Com board #1 (as shipped)

CPU board #2

Echlin's Boot floppy w/ MS-DOS 3.30

CONFIG.SYS none

AUTOEXEC.BAT PROMPT date = d_time = tg

Test method:

Record time from external clock (wristwatch)

Power on system

Record date and time as displayed on monitor

Leave on for indeterminate amount of time

Power off

Leave off for indeterminate amount of time

Repeat

Day	Cycle	Actual Date	On	Off	Display Date	Time	Time on	Time off
1	1	11/25/98	19:05		11-25-2000	19:05:19.04		
		11/25/98		19:08			0:03	
2	1	11/26/98	9:01		11-26-2000	9:01:28.92		13:53
		11/26/98		10:42			1:41	
	2	11/26/98	11:51		11-26-2000	11:51:39.18		1:09
		11/26/98		21:25			9:34	
3	1	11/27/98	10:56		11-27-2000	10:56:17.34		13:31
		11/27/98		12:38			1:42	
	2	11/27/98	20:36		11-27-2000	20:35:46.99		7:58
		11/27/98		22:44			2:08	
	3	11/27/98	22:45		11-27-2000	22:45:26.26		0:01
		11/27/98		22:46			0:01	
4	1	11/28/98	8:49		11-28-2000	8:49:36.10		10:03
		11/28/98		12:34			3:45	
	2	11/28/98	18:43		11-28-2000	18:43:03.47		6:09
		11/28/98		21:10			2:27	
	3	11/28/98	21:12		11-28-2000	21:12:03.14		0:02
		11/28/98		22:07			0:55	
5	1	11/29/98	10:33		11-29-2000	10:33:07.78		12:26
		11/29/98		18:08			7:35	
	2	11/29/98	20:22		11-29-2000	20:22:20.00		2:14
		11/29/98		20:23			0:01	
6	1	11/30/98	11:43		11-30-2000	11:43:08.38		15:20
		11/30/98		13:39			1:56	
	2	11/30/98	13:39		11-30-2000	13:39:39.84		0:00
		11/30/98		14:02			0:23	
	3	11/30/98	14:02		11-30-2000	14:02:40.50		0:00
		11/30/98		16:27			2:25	
	4	11/30/98	16:32		11-30-2000	16:32:22.31		0:05
		11/30/98		17:43			1:11	
7	1	12/1/98	7:53		12-01-2000	7:53:22.19		14:10
		12/1/98		8:12			0:19	
	2	12/1/98	8:29		12-01-2000	8:29:51.41		0:17
		12/1/98		10:22			1:53	
	3	12/1/98	10:36		12-01-2000	10:36:35.73		0:14
		12/1/98		12:49			2:13	
	4	12/1/98	13:36		12-01-2000	13:36:25.02		0:00
		12/1/98		14:01			0:25	
	5	12/1/98	15:51		12-01-2000	15:51:44.05		1:50
		12/1/98		16:17			0:26	

		Actual			Display			
Day	Cycle	Date	On	Off	Date	Time	Time on	Time off
8	1	12/2/98	7:50		12-02-2000	7:51:27.34		15:33
		12/2/98		8:30			0:40	
	2	12/2/98	9:40		12-02-2000	9:40:32.10		1:10
		12/2/98		14:00			4:20	
	3	12/2/98	14:51		12-02-2000	14:51:06.39		0:51
		12/2/98		15:14			0:23	
9	1	12/3/98	8:13		12-03-2000	8:13:25.93		16:59
		12/3/98		10:45			2:32	
	2	12/3/98	10:49		12-03-2000	10:49:56.16		0:04
		12/3/98		11:08			0:19	
	3	12/3/98	14:21		12-03-2000	14:22:06.08		3:13
		12/3/98		15:22			1:01	
	4	12/3/98	15:39		12-03-2000	15:39:00.31		0:17
		12/3/98		16:40			1:01	
10	1	12/4/98	8:39		12-04-2000	8:39:17.36		15:59
		12/4/98		9:30			0:51	
	2	12/4/98	9:31		12-04-2000	9:31:01.25		0:01
		12/4/98		15:34			6:03	
	3	12/4/98	15:35		12-04-2000	15:35:19.51		0:01
		12/4/98		15:39			0:04	
11	1	12/7/98	11:40		12-07-2000	11:40:55.53		20:01
		12/7/98		13:53			2:13	
	2	12/7/98	16:00		12-07-2000	16:00:15.32		2:07
		12/7/98		17:04			1:04	
12	1	12/8/98	8:42		12-08-2000	8:42:32.35		15:38
		12/8/98		12:01			3:19	
13	1	12/14/98	8:52		12-14-2000	8:52:39.01		20:51
		12/14/98		14:05			5:13	
14	1	12/15/98	10:03		12-15-2000	10:03:54.51		19:58
		12/15/98		10:58			0:55	
	2	12/15/98	11:30		12-15-2000	11:30:20.68		0:32
		12/15/98		16:15			4:45	
	3	12/15/98	16:18		12-15-2000	16:18:19.53		0:03
		12/15/98		17:10			0:52	
15	1	12/16/98	9:44		12-16-2000	9:44:37.94		16:34
		12/16/98		10:14			0:30	
	2	12/16/98	10:20		12-16-2000	10:21:00.24		0:06
		12/16/98		10:54			0:34	
	3	12/16/98	10:55		12-16-2000	10:55:32.47		0:01
		12/16/98		13:42			2:47	
	4	12/16/98	13:42		12-16-2000	13:42:32.91		0:00
		12/16/98		17:27			3:45	
16	1	1/4/99	8:00		01-04-2001	8:00:30.17		14:33
		1/4/99		8:02			0:02	

Appendix D: SSM/TSM Conversion Anomaly

Within standard AT architecture there are approximately 18.2 timer ticks per second (Crystal oscillator frequency of 14.31818 MHz divided by 12, then divided again by 65536.) The MS-DOS routine for converting seconds-since-midnight (SSM) to ticks-since-midnight (TSM) does the following math:

TSM = SSM * 100 * 59659 / 65536 / 5

The above DOS algorithm therefore calculates 18.20648193 timer ticks per second. This DOS algorithm is not documented. Therefore, other programs that want to convert from seconds to ticks might use a different conversion algorithm and come up with a slightly different value. The Zenith BIOS in Mr. Echlin's machine converts from seconds to timer ticks by doing the following math:

TSM = SSM * 182 / 10

The above Zenith BIOS algorithm calculates 18.2 timer ticks per second and will therefore get slightly different answers than DOS gets. The BIOS will get a 0.036% smaller number of ticks than DOS gets, which means that it will be off by about 1 second for every 2809 seconds (46 minutes 49 seconds) that are converted. Thus, there will be a discrepancy of 1 second at 12:47 AM, two seconds at 1:34 AM, and so on up to 30.76 seconds just before midnight. Note that versions of DOS based on MS-DOS (such as IBM PC-DOS and Compaq DOS) all use the same algorithm as MS-DOS. However, DR-DOS converts by using the same algorithm as the Zenith BIOS. We believe the algorithm for converting seconds to ticks used by many (if not all) BIOS's is slightly different than the algorithm used by DOS.

Appendix E: Bug in Zenith INT 1A handler

We found one bug in the interrupt INT 1A handler on Mr. Echlin's Zenith. Both functions, function 2 (Get RTC Time) and function 4 (Get RTC Date) exercise the same code path that exhibits this bug. Prior to reading the RTC, the interrupt handler checks that the RTC is working by reading CMOS index 0x0E (the CMOS Diagnostic Status Byte) and checking if any of the bits related to RTC malfunction are set:

Bit 7 - When set (1) indicates clock has lost power

Bit 6 - (1) indicates incorrect checksum

Bit 2 - (1) indicates that time is invalid

If all of those bits are clear, then the BIOS waits for the UIP to be clear and then reads the RTC indexes to get the current time (or date). If any of those bits are set, then the BIOS does not wait for the UIP to be clear, but reads the RTC indexes anyway and returns success to the caller.

The fact that the BIOS return success even when it thinks the RTC is malfunctioning is a bug. This would normally be a problem only if any of the bits in the CMOS Diagnostic Status Byte were set, which should not happen (and which does not happen in any of our tests). However, the BIOS checks the CMOS Diagnostic Status Byte with interrupts enabled. There is a two instruction window between when the BIOS selects CMOS index 0E as the index it wants to read and the instruction where it reads the Diagnostic Status Byte. If a hardware interrupt occurred during that two instruction window, and if the hardware interrupt handler accesses the CMOS, then the CMOS index that the BIOS reads upon return from the hardware interrupt could be something other than index 0E. If the value read by the BIOS from this index has any of the three bits on, then the BIOS could decide to skip the code which checks for the UIP bit being clear. If it then happens to read the RTC while the RTC is updating, the BIOS could return an invalid time (or date).

The two instruction window lasts for about 1 microsecond on the 12 MHz 286 in Mr. Echlin's Zenith. As timer interrupts usually occur about once every 55,000 microseconds (18.2 times per second), a timer interrupt would only hit that window about once every 55,000 calls to the BIOS interrupt 1A handler. Even if that occurs, the problem would only occur if the BIOS code which subsequently reads the RTC is executed while the RTC is updating, which should happen less than once every 5,000 times that the UIP is not checked. Also, no BIOS or DOS hardware interrupt handler accesses the CMOS, and the BIOS bug only shows up if the CMOS index port is changed by the hardware interrupt handler. Therefore, the bug would only show up if a piece of software with an interrupt handler that accessed CMOS was executing in conjunction with a piece of software that called the BIOS to get the time (or date). Even then it would show up less than once in every 2.75 million calls to the BIOS to get the time (or date).

This Zenith BIOS bug is not related to year 2000 issues, and we do not believe that this Zenith BIOS bug explains any of the Crouch-Echlin Effects that have been reported by Mr. Echlin and others.

Appendix F: Technical Descriptions of RTC/BIOS/DOS programming

Description of RTC Programming

The Real Time Clock (RTC) is implemented as a part of the RTC/CMOS controller chip and is read from and written to via I/O ports 0070 and 0071. The RTC returns the current seconds, minutes and hours via indexes 0, 2, and 4, respectively. The RTC returns the current day of month, month, and year (last two digits) via indexes 7, 8, and 9, respectively. The CMOS memory is used to store various system configuration parameters (such as type of floppy disk installed). The CMOS memory also contains a location for storing the century value. The century index is 032 on most machines, but is 037 on older IBM PS/2 models. The century index is not hooked to the RTC, but must instead be updated by software or firmware.

All the values in the RTC are stored in binary-coded decimal (BCD), rather than hexadecimal (hex). For example, a value of 10 would be stored as BCD 10, which is hex 16. As programs typically manipulate hex values, not BCD values, programs that interact with the RTC need to convert from hex to BCD and back. This conversion is rather straightforward when dealing with one-byte BCD numbers.

To program the RTC/CMOS controller, a program must first identify which index it wants to read or write by OUT-ing the index value to I/O port 0070. The program can then read the value of the selected index by IN-ing from I/O port 0071 or can write a new value to the selected index by OUT-ing to I/O port 0071.

The RTC specifications indicate that the RTC cannot be programmed reliably while the RTC is updating itself (i.e. while the RTC is busy updating its registers once each second). Prior to reading a value from or writing a value to the RTC, a program is supposed to check if the RTC's "update-in-progress" flag is set, and should refrain from programming the RTC while that flag is set. The RTC specification states that the "update-in-progress" flag will be set 244 microseconds before the update actually starts, so a program can safely execute 244 microseconds worth of code between the time that it verifies that the "update-in-progress" flag is clear and the time that it finishes programming the RTC.

Description of BIOS Time/Date support

At POST time, the BIOS reads the RTC to get the current time, converts the time from hours/minutes/seconds to "timer ticks since midnight" and stores that value in the BIOS data area in RAM memory.

In its timer interrupt handler (INT 08), the BIOS increments the "timer ticks since midnight" value in the BIOS data area (BDA). If the value reaches the number of timer ticks in a day, then the BIOS will reset the "ticks since midnight" to zero and will note (in the byte at offset x70 in the BDA) that a day has rolled over.

The BIOS supports six interrupt INT 1A calls (AH = 0, AH = 1, AH = 2, AH = 3, AH = 4, AH = 5) which get or set the system date or time. There are other BIOS calls which read/write the RTC for other reasons (INT 1A function 6 through 0xF), but these functions are not used by DOS and probably not used by Windows or any popular PC software.

The BIOS function to get the system time (INT 1A, AH = 0) returns the number of 18 Hz timer ticks since midnight by getting that value out of the BDA.

The BIOS function to set the system time (INT 1A, AH = 1) is passed the number of 18 Hz timer ticks since midnight and sets that value into the BDA.

The BIOS function to get the RTC time (INT 1A, AH = 2) returns the hour/minutes/seconds in binary-coded-decimal (BCD) and the daylight savings flag by reading those values from the RTC. This function may return an error if the RTC was updating or malfunctioning when it was called.

The BIOS function to set the RTC time (INT 1A, AH = 3) is passed the hours/minutes/seconds in BCD and the daylight savings flag and writes those values to the RTC. The function may not return an error, so it must retry if the RTC was updating when it was called.

The BIOS function to get the RTC date (INT 1A, AH = 4) returns the century/year/month/day by reading those values from the RTC. This function may return an error if the RTC was updating or malfunctioning when it was called.

The BIOS function to set the RTC date (INT 1A, AH = 5) is passed the century/year/month/day and writes those values to the RTC. The function may not return an error, so it must retry if the RTC was updating when it was called.

Description of DOS kernel Time/Date support

During its initialization, the DOS kernel programs the RTC to set Status Register A and Status Register B to system default values. At no other time does DOS interact directly with the RTC.

During its initialization, the DOS kernel asks the BIOS for the current RTC date, and converts the result into a "days since January 1st 1980" variable. DOS updates this internal date variable when it calls the BIOS' Get System Time call and sees that a midnight has passed since its last call.

The DOS kernel supports four interrupt INT 21 calls (AH = 2A, AH = 2B, AH = 2C, and AH = 2D) which get or set the system date or time.

The DOS kernel function to get the time (INT 21, AH = 0x2C) calls the BIOS (INT 1A, AH = 0) to get the number of 18 Hz timer ticks since midnight (stored in the BIOS data area). Then it performs some math on those numbers so it can return the hours/minutes/seconds/centi-seconds.

The DOS kernel function to set the time (INT 21, AH = 0x2D) is given the hours/minutes/seconds/centi-seconds. It asks the BIOS for the current RTC hours/minutes/seconds/daylight-savings-flag (INT 1A, AH = 2) and passes the new times and the old daylight-savings-flag to set the RTC (INT 1A, AH = 3). It then does some math so it can pass the 18 Hz timer ticks since midnight to the BIOS function (INT 1A, AH = 1) which sets those values in the BIOS data area.

The DOS kernel function to get the date (INT 21, AH = 0x2A) calculates the year/month/day based upon its internal "days since January 1st 1980" variable. It also calculates (and returns) the day of the week.

The DOS kernel function to set the date (INT 21, AH = 0x2B) is given the year/month/day and divides the year by 100 and then passes the century/year/month/day to the BIOS to set the RTC date (interrupt 0x1A, AH = 5) and updates its internal date variable.

The DOS kernel automatically tracks certain dates and times associated with disk files (such as the date and time that the file was last modified). DOS uses its internal variables to get the current date and ask the BIOS for the current "ticks-since-midnight" value to get the current time. It then converts the date to a bit-encoded 16-bit value and the time to a bit-encoded 16-bit value. These conversions have no year 2000 problems.

The DOS kernel supports various calls to get and set certain dates and times associated with disk files (such as the date and time that the file was last modified). These calls convert between year/month/day or hour/minute/second and bit-encoded 16-bit values of the date or time. These conversions are independent of RTC and BIOS issues and have no year 2000 problems.

Description of COMMAND.COM and Time/Date Algorithms

COMMAND.COM does not interact directly with the RTC or BIOS, but instead calls the DOS kernel to get and set dates and times.

COMMAND.COM calls DOS to get the current time when the user enters the TIME command or when the command prompt includes the current time.

COMMAND.COM calls DOS to get the current date when the user enters the DATE command or when the command prompt includes the current date.

COMMAND.COM calls DOS to set the current time when the user enters a new time using the TIME command. COMMAND.COM converts the time string typed by the user into hour/minute/second values to pass to DOS. Although COMMAND.COM does not bounds check the time entered, the DOS kernel will reject any invalid time (e.g. 8:61) and COMMAND.COM will report that the time entered was invalid.

COMMAND.COM calls DOS to set the current date when the user enters a new date using the DATE command. COMMAND.COM converts the date string typed by the user into year/month/day values to pass to DOS. The DATE command accepts 4-digit year values, which it accepts without bounds checking. The DATE command also accepts 2-digit year values, and assumes that the user is entering a 20th century year (i.e. it tells DOS to set the year to 19xx where "xx" was the 2-digit year the user typed). Although COMMAND.COM does not bounds check the date, the DOS kernel will reject any dates before 1980 or after 2099 as well as any invalid date (e.g. February 30th) and COMMAND.COM will report that the date entered was invalid.

COMMAND.COM displays dates and times associated with files (such as the date and time that the file was last modified).